Name aliases and UTF-16 encoding scheme are inconsistent with the Unicode Standard

Document No.	**N5168**	Date	2021-11-23

Reply To	Peter Brett **pbrett@cadence.com**
	Tom Honermann tom@honermann.net

Introduction

ISO/IEC 14822 “The Programming Language C++” has a normative reference to ISO/IEC 10646 “Universal Coded Character Set”.

Because C++ is an ISO standard, the specification is required to normatively reference ISO 10646 rather than the Unicode Standard. Many implementers of C++, however, have requirements from customers and end users to conform to the Unicode Standard.

During our work in JTC1/TC22/WG21, this requirement to consider both standards has helped us to find some issues that we think may be defects in ISO/IEC 10646:2020.

Name aliases diverge from the Unicode Standard

The Unicode Standard 14.0.0 divides name aliases into 5 types named ‘correction’, ‘control’, ‘alternate’, ‘figment’ and ‘abbreviation’, but ISO 10646 does not reflect this partitioning. The name aliases appear incomplete, relative to those provided by `NameAliases.txt` in the UCD.

For example, ISO 10646 specifies “NO BREAK HERE” as an informative alias for U+0083, but omits the “NBH” abbreviation present in `NameAliases.txt`.

The aliases present for U+0083 in the UCD are:

```
NamesList.txt       "NO BREAK HERE"
NameAliases.txt     "NO BREAK HERE" (control)
NameAliases.txt     "NBH" (abbreviation)
```

ISO 10646 contains (the ‘=’ introducer indicates an informative alias):

```
<control>
= NO BREAK HERE
```

In ISO 10646 section 34.3, “Character names list’, the normative name aliases preceded by ‘※’ appear to correspond to UCD `NameAliases.txt` aliases of type ‘correction’, but this does not appear to be explicitly stated by any normative or non-normative text.

Both these issues can cause divergence from the Unicode Standard by ISO standards that normatively reference ISO 10646, e.g. by only being able to support a subset of UCD name aliases in facilities that accept a character name or name alias.

Recommended resolution: fully synchronize the name aliases from the UCD `NameAliases.txt` into ISO 10646. Add a non-normative note that states that ‘※’ aliases represent corrections.

Control characters have no normative names

The authors of programming language standards like ISO/IEC 14882 “The Programming Language C++” wish to be able to name control characters in their lexical specifications (for example, U+000A LINE FEED).

In the Unicode Standard, names for control characters are provided normatively via `NameAliases.txt` in the UCD.

However, in ISO 10646, control characters have no normative names. There is a non-normative “NOTE 2” in section 12, “Use of control functions within the UCS”, which lists names for control characters. Unfortunately, because this is non-normative it cannot be used as a normative reference for naming by other ISO standards.

Other ISO standards which wish to reference Unicode control characters by name must therefore copy the table from “NOTE 2” in section 12, or unfortunately invent their own names for control characters.

Recommended resolution: provide the list of suggested control character names as normative text rather than in a non-normative note.

Allow ‘higher-level protocol’ to determine UTF-16 byte order

The Unicode Standard section 3.10 says (emphasis added):

> The UTF-16 encoding scheme may or may not begin with a BOM. However, when there is no BOM, **and in the absence of a higher-level protocol,** the byte order of the UTF-16 encoding scheme is big-endian.

ISO 10646 section 11.5, “UTF-16”, says:

> The UTF-16 encoding scheme serializes a UTF-16 CC-data-element by ordering octets in a way that either the less significant octet precedes or follows the more significant octet.
>
> In the UTF-16 encoding scheme, the initial signature read as <FE FF> indicates that the more significant octet precedes the less significant octet, and <FF FE> the reverse. The signature is not part of the textual data.
>
> In the absence of a signature, the octet order of the UTF-16 encoding scheme is that the more significant octet precedes the less significant octet.

Implementations conforming to the Unicode Standard may use a “higher-level protocol” (e.g. out-of-band data or non-text protocol fields) to determine the byte order of data encoded with the UTF-16 encoding scheme. However, applications conforming to ISO 10646 must not. If there is no BOM, then the byte order must always be treated as big-endian even in the presence of out-of-band data to the contrary.

Recommended resolution: update section 11.5:

> In the absence of a signature or a higher-level protocol, the octet order of the UTF-16 encoding scheme is that the more significant octet precedes the less significant octet.